Voxelization of solids using simplicial coverings

Antonio J. Rueda, Rafael J. Segura, Francisco R. Feito, Juan Ruiz de Miras, Carlos Ogáyar
Universidad de Jaén
Escuela Politécnica Superior
Avda. Madrid, 35
Spain (E) 23071, Jaén
{ajrueda,rsegura,ffeito,demiras,cogayar}@ujaen.es

ABSTRACT

Rasterization of polygons in 2D is a well known problem, existing several optimal solutions to solve it. The extension of this problem to 3D is more difficult and most existing solutions are designed to obtain a voxelization of the solid. In this paper a new approach to rasterize and voxelize solids in 3D is presented. The described algorithms are very simple, general and robust. The 3D algorithm is valid to be used in the new 3D displays, and it can also be used to voxelize solids delimited by planar faces (with or without holes, manifold or non-manifold). The proposed methods are very suitable for an implementation in graphic hardware rendering system, because it does not use any additional data structure or complex operation.

Keywords

Geometric Modeling, Solid Modeling, Voxelization, 3D Rasterization, 3D Displays.

1. INTRODUCTION

Rasterization of polygons in a 2D visualization display is one of the basic and most common operations in any graphic system. As a very desirable feature, this must be able to handle any kind of polygon, including non-convex, holed or non-manifold polygons. Two approaches have been traditionally used for this purpose: rasterize the polygon as it is using an scanline method, or decompose the polygon into triangles that can be rasterized very efficiently in most current graphic systems. The scanline polygon fill algorithm [Fol94] is a popular approach for the rasterization of polygons. The basic idea beyond it is to perform a line sweep of the polygon, using a list of the active edges intersected by the current scanline to determine which pixels must be set in the framebuffer. By far, the most common approach is based on a previous tessellation of the polygon and the rasterization the resulting triangles. Its motivation is to take advantage of the efficient hardware triangle rasterization available in most current graphic systems. Nevertheless, general polygon triangulation is a hard problem [Bern92], and a reasonable solution can only be done in $O(n \cdot log\ n)$. The complexity of these solutions mades very unpractical an implementation in hardware. For instance, OpenGL is only able to display convex polygons and triangles, but it includes an efficient software tessellator that can be used to triangulate complex polygons.

In the last years, the advances towards the construction of cheap 3D displays have been very important [Blundell00]. This kind of displays can be grouped basically in two categories [Pastoor00]: 3D displays based on stereoscopic images, in which the three dimensional environment is simulated by compositing several images on 2D displays; and real 3D displays, in which a real 3D image is displayed, not depending on the position of the observer. On the second category, the most promising ones are the crossed-beam displays (CBD) [Ebert99], based on the excitation of ions by two lasers of different wavelength. Other interesting

WSCG SHORT Communication papers proceeding
WSCG 2004, February 2-6, 2003, Plzen, Czech Republic.

displays are those based on holography, although the subjacent mechanism is similar to CBD.

Apart from the specific technical problems, another important problem in 3D visualization is the absence of algorithms and libraries to help the programmer to manage the large data structures needed to visualize them in a easy and quick way. In this sense, some intents to construct libraries to these type of data has been made in the last years; the main proposal has been to extend some of the well known libraries to 2D Displays (as OpenGL, Java3D) by adding new functionality. In the case of rasterizing volumetric solids, there are few algorithms to solve the problem directly; instead of it, the algorithms used are extension of voxelization algorithms.

The simplest approach to perform a solid voxelization consists of testing the inclusion of the center of every voxel in the solid. The inclusion of the center of a voxel can be calculated using Jordan Curve Theorem. The main drawback of this method is its low performance (for each point to be tested all polygons must be checked for intersection, unless a preprocessing stage is done). In addition, this process only works with solids.

Another straightforward way to voxelize a solid is based on a scanline algorithm. This approach works by casting rays following an axis aligned direction (e.g. x-axis direction). Each ray is cast for testing the list of intersections for a given span (e.g. spans of the voxel space in x-direction); then, the list of intersections is used in a scanline algorithm way to rasterize a 3D span of the voxel space.

Huang [Huang98] describes a method for voxelizing planar objects which provides topological comformity through geometric measurements. This method eliminates common voxelization artifacts at edges and vertices. It is based on 3D discrete spaces and separability, that is: to voxelize a plane (and a polygon) two parallel planes are built, so the plane to be voxelized lies between then (all planes are parallel). This method works fine, but it does not allow the voxelization of the inner part of the solid.

Sramek [Sramek99] introduces the Voxelization Model (V-model), which is an alias-free voxelization method for geometric objects. The V-model of an object represents it in a three-dimensional continuous space by a trivariate density function. This function is sampled during the voxelization and the resulting values are stored in a volume buffer. Several filtering and interpolation methods can be applied to the surface density profile. This method allows an alias-free discrete representation of an object, but it does not take care of the inner part of the solid.

The method presented by Haumont [Haumont02] converts complete polygonal scenes into voxelized representations. It stores the status (in/out) of the volumetric space areas in the cells of an octree. First, the algorithm looks for a point in the scene for which the status can be determined; second, the status is propagated to the surrounding visible cells. This two steps are repeated until the status of all the cells in the octree is determined. The advantage of this method is its robustness, it can successfully handle issues like cracks, holes, interpenetrating meshes and overlapping geometries. The drawbacks of this technique are its noticeable slow performance and its high memory requirement.

Jones [Jones96] presents a method which voxelizes a model using a point to triangle distance function. With this approach, each voxel on the grid is treated as a point, and its distance to each triangle of the model is calculated. There are several optimizations to enhance the performance, but in general it is a slow method. Like other approaches, it does not take care of the interior of the solid.

In this paper, a method to rasterize polygons and its extension to 3D solids are presented. These methods do not require tessellations, sortings or complex data structures, so they can be easily implemented in hardware. The second method is designed specifically to rasterize solids in real 3D displays, and can be easily included in 3D libraries. The method can be also used to voxelizate solids delimited by planar faces. For other type of solids, the algorithm can be also used: in order to do it, the process is similar than the one used to draw it using some of the libraries (OpenGL, DirectX, ...): first, the solid is approximated using triangles, and then these triangles are drawn. In our case, the triangles will be used to obtain the rasterization of the solid.

2. THEORETICAL FOUNDATIONS

The following definitions are the theoretical basis of the Solid Modelling by simplicial coverings [Segura01], and let us obtain a theorem to represent the solids.

Definition 1. Let $x \in \Re$. The function *sign(x)* is defined as

$$sign(x) = \begin{cases} 1 & if \quad x > 0 \\ 0 & if \quad x = 0 \\ -1 & if \quad x < 0 \end{cases}$$

Definition 2. Let *T=(A,B,C)* be a triangle; the signed area of *T* (denoted by *[T]*) is defined as

$$[T] = sign\left(\frac{1}{2}\cdot\begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}\right)$$

Let four points be $A,B,C,D \in R^3$. The signed volume of the tetrahedron of vertices *D*, *A*, *B*, and *C*, denoted by *[DABC]*, is defined as

$$[DABC] = sign\left(\frac{1}{6}\cdot\begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{vmatrix}\right)$$

It is said that a triangle/tetrahedron is a positive triangle/tetrahedron if its signed area/volume is positive It can be easily proved that a tetrahedron has positive orientation (that is, the remaining vertices are ordered counterclockwise with respect to one vertex) if the signed volume is positive.

Definition 3. The signed volume of a pyramid *P* with vertex *V* and base $F(V_1V_2\ldots V_n)$, is denoted by [P] and is computed as

$$[P] = \sum_{i=1}^{n} [VQV_iV_{i\oplus 1}]$$

being *Q* an arbitrary point laying on plane defined by F. If the vertex of the pyramid coincides with the origin of co-ordinates it is said to be an *original pyramid*.

Theorem 1.[Fei97a]} *Generator System*. Let *S* be a solid with faces $F_1F_2...F_m$, given in consistent orientation (the normal vector goes outside the solid). Then

$$S = \sum_{i=1}^{m} [P_i] \cdot P_i$$

where P_i represents the original pyramid obtained by joining the face F_i with the origin of co-ordinates.

Proof. See [Fei97a]

In the case of 2D polygons, equation of theorem 1 is also valid, although instead of pyramids we have triangles OF_i defined between the origin and an edge F_i of the polygon and the sign of the triangles is defined by their signed area.

Instead of using pyramids, we can use tetrahedra; this will allow us a simplification in the computations [Fei97a]. As it can be seen, the pyramids do not have to be disjoint. This will allow us to work with coverings of the solids, instead of disjoint partitions of them. The main advantage of this approach is that the covering can be obtained in a very simple way with an linear algorithm, keeping the initial representation of the solid (a vertex-edge-face graph). Another advantage is that it is not neeeded to store the triangulation of the solid; it is only needed to know the edges of the solid and an arbitrary point, and therefore, there is no additional information to store.

Definition 5. Let *P* be a polygon, the *covering of the P*, denoted *as* C_p, is the set of triangles obtained by joining an arbitrary point of the plane of P (by example, the centroid of the polygon) with every edge of the polygon.

Analogy, let *S* be a solid, *the covering of S*, denoted as C_s, is the set of tetrahedral obtained by joining every triangle of the covering of every face of S with an arbitrary point (by example, the centroid of the solid).

Theorem 2. [Fei97b]. Let *Q* be a point, and *S* be a solid *(*a polygon in 2D*)*. Then *Q* is inside *S* if

$$\sum_i sign(Q,T_i)\cdot[T_i] = 1$$

where $T_i \in C_s$, $[T_i]$ is the signed volume (or signed area in 2D) of the simplex, and the function $sign(Q,T_i)$ returns the signed volume (or area in 2D) of the simplex formed by point *Q* and simplex T_i (an edge in 2D or a face in 3D).

Corollary 1. Let Q be a point inside solid *S*. Then $\exists_1 T_i \in C_s$, $[T_i]>0$, with *Q* included in T_i.

Proof. Trivially, it can be seen that, when the inclusion of a point in a solid is computed, we only use algebraic adding operations. So, at any moment it must be true that the sign of T_i is positive to obtain a positive result. Also, it is trivial to prove that the points of the solid included in negative T_i are also included in, at least, two positive T_j, because the result must be positive.

Corollary 2. Let Q be a point be and an original positive tetrahedron *T=(OABC)*. Then *Q* is inside *T* if

$$[OABQ] \geq 0 \wedge [OBCQ] \geq 0 \wedge [OCAQ] \geq 0$$

Proof. Trivially this corollary can be proved by considering the particular case of tetrahedra for solids of the theorem 2. In the case of negative tetrahedra, the sense of the comparison must be changed.

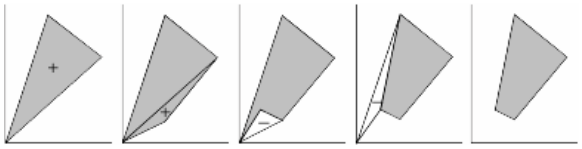

Figure 1. Polygon construction through additions and subtractions of triangles

3. 3D RASTERIZATION

Theorem 1 and 2 give us a method to construct solids in 2D and 3D. Figure 1 illustrates the construction of a 2D solid by the method shown in def. 4. An rasterization method of complex polygon based on these theorems is shown in [Rueda02]. The polygon and triangle algorithms are resumed in figures 2 and 3. The algorithms uses an intermediate buffer, called P-Buffer (Presence Buffer) in order to store each simplex of the covering of the polygon.

The 3D rasterization is similar to the 2D rasterization, but in this case we have a 3D framebuffer and a 3D P-Buffer.

1. Compute the minimal bounding box and the centroid of the polygon.
2. Construct a triangle between the origin vertex (centroid) and one edge of the polygon.
3. Rasterize the triangle in the P-buffer, flipping all positions covered by it.
4. Return to step 2 until there are not any edges left.
5. Transfer all positions from the P-buffer in the minimal bounding box of the polygon and presence values equal to 1 to the framebuffer, applying its corresponding color.

Figure 2. Rasterization algorithm for polygons

Each position of the P-Buffer stores the presence value of the voxels, i.e. if the voxel belongs or not to the solid. The representation of this presence value is a bit, which is sucessively flipped during the rasterization when a tetrahedron covers it.

Once the rasterization of the solid has been completed, the information stored on the P-buffer is transferred to the framebuffer, applying its corresponding colour. The colour of every voxel depends on the properties of the solid. If we consider only homogeneous solids, the color of every voxel is always the same.

1. Sort the vertices *ABC* by their *y* coordinates.
2. *xl=xr=C.x* ; *y=C.y*
   // *C* is the vertex with the least *y* coordinate
3. Initialize *il* and *ir* to the slopes of the segments $\overline{CA}$ and $\overline{CB}$ respectively. If *ir* is less than *il*, swap *ir* and *il*, and *xr* and *xl*.
4. If *y* reaches *B.y* then set *ir* or *il* (if it was swapped in step 2) to the slope of the segment $\overline{BA}$.
5. Add or subtract the sign of the triangle to each position of the P-Buffer from (*xl*,*y*) to (*xr*,*y*).
6. *y++*; *xl+=il*; *xr+=ir*
7. If *y A.y*, return to step 4.

Figure 3. Rasterization algorithms for triangles

The application of the color to every voxel is not an easy process because requires a separate computation of the texture of the voxel. The detailed steps of the 3D rasterization are shown in algorithm of figure 4.

1. For every triangle of the covering of each face of the solid, construct tetrahedra by joining the triangle with the origin of co-ordinates.
2. Rasterize every tetrahedra obtained in previous step in the P-Buffer, changing the value of the positions covered by the tetrahedron.
3. Transfer all positions with presence value being equal than 1 to the frame buffer, by applying a function such as, given a point of the solid, it returns the corresponding colour of the solid in that point. The definition of this function depends on every solid.

Figure 4. Algorithm for voxelization of solids

Rasterizing Tetrahedra

As it has just been shown, the kernel of the previous algorithm is the rasterization of a tetrahedron *OABC*. In order to solve it, we propose an extension of the ordinary scan-line trough different slices of the tetrahedron. The proposed method is given by four steps:

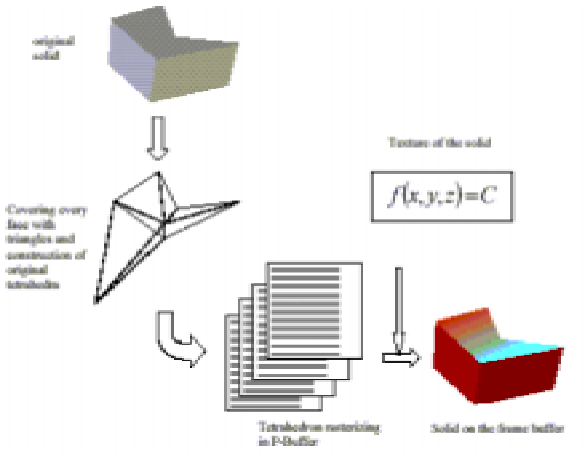

Figure 5. Process of rasterization of a solid

1. Initially it is necessary to determine which plane will be used to make the sweeping. In order to do it, we consider the maximum absolute value of the co-ordinates of the normal vector of the base of the tetrahedron. This vector will be the same for every triangle of a face of the solid (see fig. 5). We will suppose in the algorithm that the rasterization must be done respect the *Y* direction. If the maximum is negative, then the tetrahedron will be inverted with respect to the y coordinate; at the end of the process, the obtained voxel (x_i,y_i,z_i) will be also inverted to $(x_i,-y_i,z_i)$.

2. On the second step, we search for the vertex of the tetrahedron with maximum value in coordinate y. In figure 5 the point with maximum y coordinate is vertex *A*. Once the maximum is obtained, we use the equation of the edges to compute the increments of the triangles in every slice. It is necessary to do it for the edges *AB*, *AC*, *BC*, *OA*, *OB* and *OC*. These increments are computed as :

$$\frac{x-A_x}{B_x-A_x}=\frac{y-A_y}{B_y-A_y}=\frac{z-A_z}{B_z-A_z}$$

then

$$x=A_x+(y-A_y)\frac{B_x-A_x}{B_y-A_y}=A_x+(y-A_y)\frac{1}{m^x_{AB}}$$

$$z=A_z+(y-A_y)\frac{B_z-A_z}{B_y-A_y}=A_z+(y-A_y)\frac{1}{m^z_{AB}}$$

Also, for edge *OA*,

$$\frac{x}{A_x}=\frac{y}{A_y}=\frac{z}{A_z}$$

and then

$$x=y\cdot\frac{A_x}{A_y}=y\cdot\frac{1}{m^x_{OA}},\ z=y\cdot\frac{A_z}{A_y}=y\cdot\frac{1}{m^z_{OA}}$$

In order to simplify the notation, it is called m_{AB} as the pair (m^x_{AB}, m^z_{AB}). Equally, we will use m_{AC}, m_{BC}, m_{OA}, m_{OB} and m_{OC}. It is important to note that the extremes of the triangle in every step can be computed using an incremental approach. So, for the edge OA, in the step i+1,

$$\frac{x_{i+1}}{A_x}=\frac{y_{i+1}}{A_x}=\frac{y_i+1}{A_x}\Rightarrow x_{i+1}=x_i+\frac{1}{m^x_{OA}}$$

$$\frac{z_{i+1}}{A_z}=\frac{y_{i+1}}{A_z}=\frac{y_i+1}{A_z}\Rightarrow z_{i+1}=z_i+\frac{1}{m^z_{OA}}$$

For the other edges, a similar approach is used. We can also use the incremental approach used in the algorithm to draw lines [Fol94], in order to avoid the use of divisions.

3. On the third step, we have to rasterize the triangle obtained as intersection of the tetrahedron with the plane $Y=y_i$. We consider only the slices between the maximum *y* coordinate of the base of the tetrahedron and the minimum *y* coordinate of the vertices of the tetrahedron (in our case, it will be always zero). The extremes of the slice to be considered are computed using the corresponding parameter *m*. So, if the *y* coordinate of the slice is greater than B_y, then the parameter m_{AB} will be used to update the extreme of the triangle over the edge *AB*. In other case, we must use the parameter m_{OB} to update this extreme. The vertex *C* will be updated equally. For the vertex *A*, the parameter m_{OA} will be always used.
In order to rasterize the triangle, the scan-line algorithm to fill polygon in 2D (adapted to triangles) must be used. For any pixel (x_i,z_i) inside the triangle, the corresponding voxel (x_i,y_i,z_i) must be changed from 1 to 0, or from 0 to 1. Initially, all the voxels are initialized to 0. It is important to note that when $B_i>y_i>C_i$ then the slice obtained is a polygon with four vertices (see fig.6.c). In this case, the polygon is decomposed in two triangles.

Algorithm appearing in fig.7 summarizes the process described here.

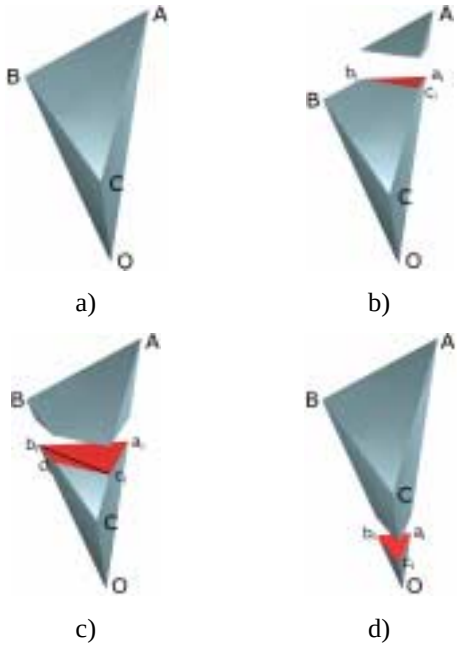

Figure 6. Process of rasterization of a tetrahedron

4. PERFORMANCE.

In order to prove the validity of the algorithm, some tests have been made using different kind of solids. Figure 9 shows some of the results obtained. As it can be seen on figure 9, the obtained images present zones with aliasing. Both in two and three dimensions, the main problem of the rasterization (voxelization) algorithms is the aliasing. This effect appears by using integer arithmetic on floating objects (polygons or solids). There are several ways to minimize (not to eliminate) this problem. Most of these techniques can be easily extended to the proposed algorithm.

In order to compare the performance of the proposed algorithm, we have made some tests using different solids and different grid resolutions. We have compared our method with Sramek's one and the one based on scanline (noted in the tables as Jordan method). The results obtained are shown in table 1. Tests have been made on a Pentium III 1GHz with 768 MB of RAM memory. The algorithm have been implemented using C++ language. The times obtained with the method based on scaline are very high, and therefore they are not considered in the study.

Initialize the matrix of voxels.
Sort ABC with respect to the *y* co-ordinate
$y=A_y$
Compute m_{AB}, m_{AC}, m_{BC}, m_{OA}, m_{OB}, m_{OC}
$A_i=A$; $m_B=m_{AB}$; $m_C=m_{AC}$
if $B_y=y$ **then** $B_i=B$; $m_B=m_{OB}$
else $m_B=m_{AB}$; $B_i=A_i+(B-A)\cdot m_B$
if $C_y=y$ **then** $C_i=C$; $m_C=m_{OC}$
else $m_C=m_{AC}$; $C_i=A_i+(C-A)\cdot m_C$
While $(y>0)$
    Rasterize2D (A_i,B_i,C_i)
    y--;
    if $(y<B_i)$ **then** $m_B=m_{OB}$
    if $(y<C_i)$ **then** $m_C=m_{OC}$
    if $(B_y>y>C_y)$ **then**
        $D_i=B+(C-B)\cdot m_{BC}$
        Rasterize2D(B_i,D_i,C_i)
    Update A_i,B_i,C_i with the corresponding increment

Figure 7. Rasterization of a tetrahedron

The algorithm proposed in figure 7 is quicker than Sramek and Jordan based ones in most of the situations. Sramek's method is quicker than our method for lower resolutions; when the number of faces of the solid grows up, then the difference decreases, but when resolution is increased, then the difference between our algorithm an the Sramek's one is higher. Figure 8 shows a comparison between both methods (we use logarithmic scale for times to show the difference in a more clear way). It is obvious that our method depends on the number of faces of the solid in a clearer way than the Sramek one; times obtained with Sramek's algorithm depend only on the resolution of the grid and the volume of the solid; our method also depends on the number of faces.

One disadvantage of Sramek's method is that it only rasterizes the boundary of the solid; our method obtain a complete voxelization of the solid in one pass, and it can be adapted easily to obtain a rasterization of the boundary of the solid. In order to do it, it is only necessary to consider in each iteration the spans obtained by joining the points B_i and C_i (or D_i and C_i when $B_y<y<C_y$), because they are the points belonging to the boundary of the solid (see fig.6 and algorithm of fig.7). In Table 2 the times obtained by the new algorithm obtaining only the rasterization of the boundary of the solid are shown. As it can be seen, times are better than the one's obtained with the Sramek's algorithm, although no antialising solutions are provided.

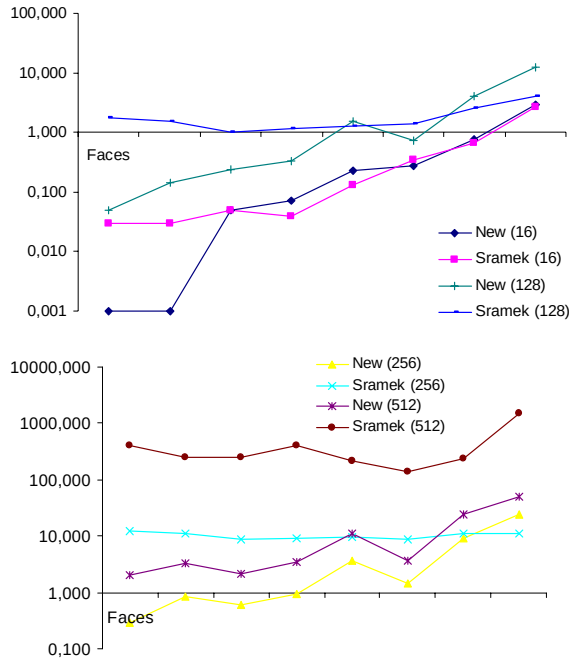

Figure 8. Comparison of times between Sramek's and the proposed algorithms

		16		128		512	
Vertices	Faces	New	Sramek	New	Sramek	New	Sramek
25	42	0,001	0,03	0,05	1,743	1,992	402,999
120	234	0,001	0,03	0,141	1,542	3,254	251,882
1849	2366	0,05	0,05	0,24	1,012	2,163	244,893
8736	2912	0,07	0,04	0,33	1,181	3,515	400,677
13025	25946	0,271	0,341	0,731	1,372	3,605	139,111
23370	46205	0,781	0,661	4,156	2,504	24,635	237,601
100250	202520	2,874	2,694	12,488	4,006	51,013	1512,128

Table 1. Comparison of times (sec.) with different resolutions

Also, Sramek's method is not suitable to be applied to heterogeneous solids, because the interior of the solids are not considered in the process of rasterization. This is an advantage of our algorithm because it can be addapted for rasterization and voxelization problems.

Vertices	Faces	16	128	512
25	42	0,008	0,017	0,699
120	234	0,017	0,068	0,886
1849	2366	0,034	0,085	0,937
8736	2912	0,051	0,119	1,038
4763	9522	0,136	0,238	1,310
13025	25946	0,393	0,478	1,446
23370	46205	0,733	0,869	2,485
100250	202520	1,512	3,234	4,715

Table 2: Times obtained by the algorithm, considering only the boundary of the solid

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an original approach for the rasterization and voxelization of 3D solids, obtained as generalization of a similar algorithm for rasterization in 2D. It is conceptually simpler than traditional methods and valid for general polyhedra.

Because of its simplicity (no complex data structures or algorithms are used) it is a very good candidate for a partial or full hardware fast implementation. It can also be easily parallelized because each tetrahedron can be rasterized in parallel by different processors, although this still needs a much deeper study.

The first line of future work is the develop of hardware implementations for these approaches. A second area of interest is their extension to handle curved-edges polygons and solids defined by curved surfaces.

6. ACKNOWLEDGEMENTS

This work has been partially granted by the Ministry of Science and Technology of Spain and the European Union by means of the ERDF funds, under the research project TIC2001-2003-C03-03

7. REFERENCES

[Bern92] Bern, M., Eppstein, D. Mesh generation and optimal triangulation, Computing in Euclidean Geometry, World Scientific, 1992.

[Blundell00] Blundell, B., Schwarz, A., Volumetric Three-Dimensional Display Systems, John Wiley, 2000.

[Ebert99] Ebert, D., Bedwell, E., Maher, S., Smoliar, L., Downing, E. Realizing 3D visualition using crossed-beam volumetric displays, Communications of the ACM, 42(8):101-107, 1999.

[Fang00]Fang, S., Chen, H., Hardware Accelerated voxelization, Computer & Graphics, 24(3):433--442, 2000.

[Fei97a] Feito, F., Torres, J.C., Boundary Represen-tation of Polyhedral Heterogeneous in the context of a Graphic Object Algebra. The Visual Computer}, Vol. 13, 64--77, 1997.

[Fei97b] Feito, F., Torres, J.C., Inclusion test in general polyhedra. Computer & Graphics, Vol. 21(1), 23--30, 1997.

[Fol94] Foley , J. e.a. Introduction to Computer Graphics, Addison Wesley, 1994.

[Haumont02] Haumont, D., Warzée, N. Complete Polygonal Scene Voxelization. Journal of Graphics Tools, 7(3) pp:27-41, 2002

[Huang98] Huang, J., Yagel, R., Filippov, V., Kurzion, Y. An accurate Method for Voxelizing Polygon Meshes. Proceedings of the IEEE symposium on Volume Visualization, pp: 119-126, 1998.

[Jones96] Jones, M.W., The Production of Volume Data from Triangular Meshes Using Voxelisation. Computer Graphics Forum. Vol. 15, No. 5, pp. 311-318. 1996.

[Pastoor00] Pastoor, S., Kiesewetter, R., 3-D Displays: A review of current technologies, DISPLAYS 17, pp. 100-110, 1997.

[Rueda02] Rueda, A.J., Segura, R.J., Ruiz, J., Feito, F.R., An Unified Approach for 2D and 3D Rasterization. Proc. of 1st Ibero-American Symposium in Computer Graphics, Guimaraes, Portugal, 2002.

[Segura01] Segura, R. J., Modelado de Sólidos mediante Recubrimientos Simpliciales, PhD., Dep. Lenguajes y Sistemas Informáticos, Universidad de Granada, 2001 (In Spanish)

[Sramek99] Sramek, M., Kaufman, A. Alias-Free Voxelization of Geometric Objects. IEEE Transactions on Visualization and Computer Graphics, Vol. 5(3), 1999.

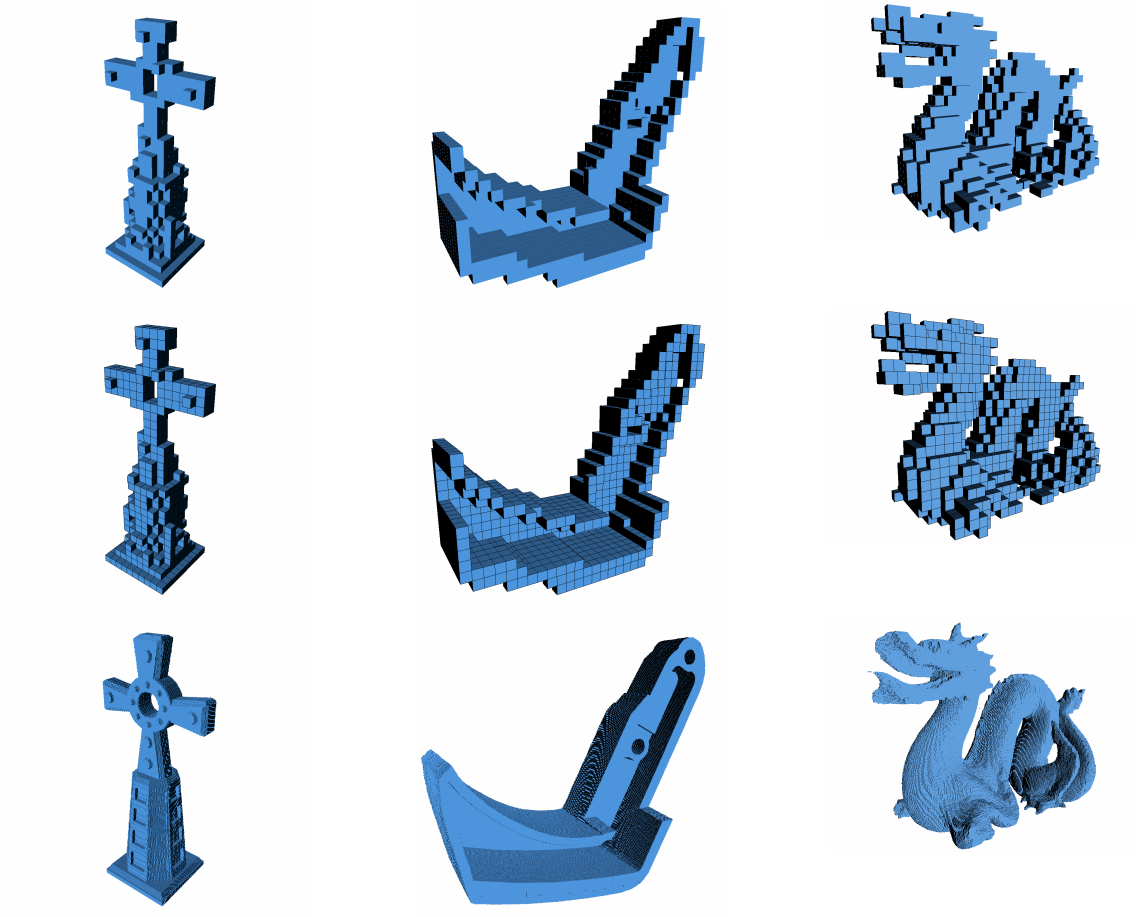

Figure 9. Voxelizing volumetric solids using different resolutions (16, 32 y 512 voxels per axis)